Applying Speculative Technique to Improve TCP Throughput over Lossy Links

Haowei Bai[*], **David Lilja**[†], **Mohammed Atiquzzaman**[‡]

[*] Honeywell Labs
3660 Technology Drive
Minneapolis, MN 55418, USA
E-mail: haowei.bai@honeywell.com
Phone: 612-951-7107
Fax: 612-951-7438

[†]Department of Electrical and Computer Engineering
University of Minnesota
200 Union St. SE
Minneapolis, MN 55455, USA
E-mail: lilja@ece.umn.edu

[‡]School of Computer Science
University of Oklahoma
Norman, OK 73019-6151, USA
E-mail: atiq@ou.edu

Abstract

The throughput degradation of Transport Control Protocol (TCP) over lossy links due to the coexistence of congestion losses and link corruption losses is very similar to the degradation of processor performance due to control hazards in CPU design. First, two types of loss events in networks with lossy links can be considered as two possibilities of a branching result (correct speculation vs. incorrect speculation) in a CPU. Secondly, both problems result in performance degradations in their application environments, i.e., penalties (in clock cycles) in a processor, and throughput degradation (in bit per second) in TCP networks. This has motivated us to apply speculative techniques (i.e., speculating on the outcome of branch predictions), used to overcome control dependencies in a processor, to TCP improvements when lossy links are involved in TCP connections. The *objective* of this paper is to propose a protocol-level speculation based TCP modification to improve its throughput over lossy links. Simulation results show that, compared to other prior research, our proposed algorithm significantly improves TCP throughput in a network with satellite links.

Keywords: Transport control protocol (TCP), Congestion control, Speculative execution, Wireless networks

Applying Speculative Technique to Improve TCP Throughput over Lossy Links

Haowei Bai
Honeywell Labs
3660 Technology Drive
Minneapolis, MN 55418
haowei.bai@honeywell.com

David Lilja
Electrical and Computer Engineering
University of Minnesota
200 Union St. SE, Minneapolis, MN 55455
lilja@ece.umn.edu

Mohammed Atiquzzaman
School of Computer Science
University of Oklahoma
Norman, OK 73019-6151
atiq@ou.edu

Abstract— **The throughput degradation of Transport Control Protocol (TCP) over lossy links due to the coexistence of congestion losses and link corruption losses is very similar to the degradation of processor performance due to control hazards in CPU design. First, two types of loss events in networks with lossy links can be considered as two possibilities of a branching result (correct speculation vs. incorrect speculation) in a CPU. Secondly, both problems result in performance degradations in their application environments, i.e., penalties (in clock cycles) in a processor, and throughput degradation (in bit per second) in TCP networks. This has motivated us to apply speculative techniques (i.e., speculating on the outcome of branch predictions), used to overcome control dependencies in a processor, to TCP improvements when lossy links are involved in TCP connections. The *objective* of this paper is to propose a protocol-level speculation based TCP modification to improve its throughput over lossy links. Simulation results show that, compared to other prior research, our proposed algorithm significantly improves TCP throughput in a network with satellite links.**

Keywords: Transport control protocol (TCP), Congestion control, Speculative execution, Wireless networks

I. Introduction

TCP was originally designed for wireline networks, where packet losses are mostly caused by network congestions. The current TCP algorithm uses either retransmission timer timing out, or receipts of three duplicated acknowledgements (ACKs) sent by receivers, to implicitly indicate loss events. However, wireless links are characterized by high error rates (see [1] for a tutorial on errors in wireless networks). In most cases, packet losses due to link corruption are more significant than congestion losses when a lossy wireless link is involved in a TCP connection. In such a case, TCP may not be able to transmit or receive at the full available bandwidth, because the TCP congestion control algorithm misinterprets link errors as congestion losses [2]. It is expected that a modification could be made which enables TCP congestion control algorithm to differentiate, and furthermore behave appropriately in the presence of both congestion and corruption losses. Significant performance improvements can be achieved if losses due to network congestion and link corruption in lossy wireless links could be appropriately differentiated [3].

Similar problems exist in computer architecture design, where control hazards prevent a processor from starting the next instruction before the correctness of branch prediction results is verified. This may cause at least one-cycle penalty which degrades the processor performance. Researchers try to overcome control dependencies in order to exploit more instruction-level parallelism by speculating on the outcome of branches and executing the program as if predictions were correct [4] [5]. Meanwhile, it is necessary to have a scheme with ability to handle incorrect speculations (e.g., flush the reorder buffer). The similarity between the degradation of processor performance due to control hazards in CPU design, and the degradation of transport-layer throughput due to the coexistence of two types of losses in network protocol design is summarized in Table I. This has motivated us to investigate the possibility of applying speculative techniques (i.e., speculating on the outcome of branch predictions), used to overcome control dependencies in a processor, to TCP congestion control when lossy links are involved in TCP connections.

TABLE I
Similarity in processor design and TCP protocol design.

	Processor Design	TCP Protocol Design
CTQ (Critical to Quality)	Execution time	Effective throughput
Problem	Control hazards degrade processor performance	Coexistence of two types of losses degrade TCP performance in wireless networks
What degrades performance	Two possible branching results	Two possible types of losses
Why degrades performance	Wasting time waiting for branching results	Wasting bandwidth responding to link corruption losses
A solution	Speculations (execute instructions as if branch predictions were always right)	Speculations (apply TCP congestion control as if all losses were due to link corruption)
Key behind speculation	Out-of-order execution; in-order commitment	Out-of-order loss differentiation; in-order packet retransmission
What if speculation was wrong	Undo or flush previous execution results	Improve speculation accuracy by minimizing congestion losses

In order to improve TCP throughput over lossy links, we propose that TCP does not decrement its congestion window size when a sender receives loss indicators (e.g., retransmission timer timing out), as if all packet losses were caused by link corruptions. This is similar to speculation techniques that computer architecture community have used to eliminate the potential clock-cycle penalties caused by branch hazards. However, unlike the speculation techniques used in processor design, if the speculations are incorrect in the case of TCP congestion control, there are no ways to ”undo” or ”flush” the actions. This is because execution results of instructions in a processor are values of pre-designed calculations, while actions of TCP congestion control algorithm are changes (increments or decrements) to the TCP sending speed.

An alternative approach to solving mispredictions is to improve the accuracy of the speculation. We propose to improve the speculation accuracy by minimizing the probability of congestion losses. This is done by optimally dimensioning the buffer of Explicit Congestion Notification (ECN) capable Random Early Detection (RED) gateways in the network. ECN [6] has been proposed by the Internet Engineering Task Force (IETF) to explicitly inform TCP senders of congestion at routers, without requiring them to wait for either a retransmission timer timeout or three duplicate ACKs. ECN has been recommended to be used in conjunction with RED [7] [8].

Preliminary work in minimizing packet losses at routers have been reported in the literature [9] [10] [11]. The first study by Liu et al. [9] used a deterministic method to mark packets. A packet is marked as CE with a probability of one if the average queue level exceeds a threshold. The study did not consider RED which uses a linear drop function and two thresholds in the buffer. The second study by Kunniyur et al. [10] did not evaluate the effect of maximum threshold on packet drops at a RED router. Bai and Atiquzzaman have reported their results of a detailed investigation on this issue in their recent paper [11].

The *objective* of this paper is to propose a new TCP algorithm based on protocol-level speculations to improve its throughput over lossy links. We call this speculation-based algorithm *SpecTCP* (Speculative TCP). In order to improve the speculation accuracy, we develop an analytical model to determine the optimal value of the RED's maximum threshold with an aim of minimizing congestion losses at RED gateways. Little research exists on tuning and optimizing RED and ECN parameters [12]. The significance of our analytical model is that the RED buffer size, and consequently the queuing delay, can be much smaller than what has been proposed earlier.

The rest of the paper is organized as follows. The proposed *SpecTCP* algorithm is presented in Section II. In order to improve the speculation accuracy, we describe our model used to minimize congestion losses and simulation-based validation in Section III. Simulation evaluation results for *SpecTCP* is presented in Section IV, followed by concluding remarks in Section V.

II. SpecTCP: Applying Speculative Techniques to Transport Layer Protocol

In this section, we describe our proposed *SpecTCP* in details. Before illustrating the principle of our proposed *SpecTCP* algorithm, we make the following assumptions:

- Our proposed algorithm is used within a WAN, an enterprise network (e.g., a private satellite network), or a network in battlefields, where it is possible to make all routers and end-systems ECN-capable.
- Our main goal in this paper is to deal with high bit error rate (BER) of lossy links. Mobility issues such as handoff and power requirements are very important in wireless networks; we are currently extending our work to cover these issues.

Performance comparison with prior algorithms (See Section IV) shows that *SpecTCP* significantly outperforms some TCP variants for lossy links which do not require ECN-capable routers and end systems.

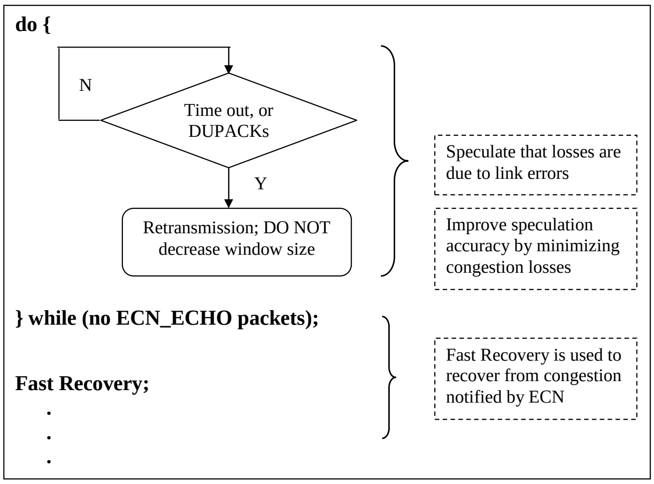

Fig. 1. *SpecTCP* software kernel.

Authors in [13] pointed out that packet losses due to queue buffer overflows is relatively infrequent when a majority of end-systems become ECN-capable and participate in TCP or other compatible congestion control mechanisms. Furthermore, if the RED threshold is appropriately selected (as described in Section III-C), we are able to minimize congestion losses. In addition, link errors become more significant compared to packet losses due to buffer overflows in wireless links. Therefore, it is reasonable to speculate that all loss events are due to links errors, unless the speculation is wrong, i.e., the congestion is explicitly reported by ECN. In this case, Fast Recovery scheme is triggered.

Figure 1 shows the kernel of our proposed *SpecTCP* algorithm at the sender's side. A *SpecTCP* sender treats retransmission timer timing out and/or three duplicate acknowledgements as indications of link errors. Most often, this is the case in a network with wireless links (packet losses due to link errors). In this case, the *SpecTCP* source does not decrease *cwnd*, even though the speculation correctness is unknown. As shown in Fig. 1, this speculation is based on the condition that no ECN_ECHO packets are received (i.e., potentially no congestion losses). Therefore, in order to improve the speculation accuracy, a scheme is required to minimize losses due to network congestion. We will discuss it in Section III. If incorrect speculation was made, i.e., the *SpecTCP* sender receives the ECN_ECHO packet sent by the receiver, the sender treats it as network congestion and triggers the Fast Recovery algorithm [14] as in the current TCP.

In *SpecTCP*, the congestion window size is appropriately controlled in the presence of either network congestion or corruption. Congestion window is halved using Fast Recovery algorithm when there is network congestion (explicitly notified by ECN_ECHO packets), and persists at the previous value in the presence of corruption. There are two mechanisms that might be applied to adjust congestion window when *SpecTCP* sender detects corruption: (i) keep *cwnd* unchanged as the previous value; (ii) use Congestion Avoidance algorithm to slowly increase *cwnd*. In our algorithm, we adopt the first mechanism — make congestion window persist in the previous value.

III. Improving Speculation Accuracy: Minimizing Congestion Losses

A. Notations

We consider a RED buffer at R1 (see Fig. 2) which is fed by multiple TCP sources. The link connecting R1 and R2 is the bottleneck link which causes congestion at R1. The sources, destinations and the RED buffer use ECN for end-to-end congestion control. The following notations will be used in our model (in Sec. III-C):

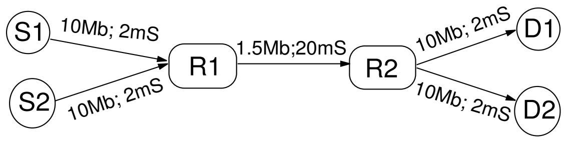

Fig. 2. Simulation topology.

- $Q(t)$: Instantaneous RED buffer size at time t.
- $\overline{Q}$, $\overline{Q}_{max}$: Average and maximum average queue sizes, respectively, at the RED buffer.
- ω: Weighting factor for calculating $\overline{Q}$.
- $p(t)$: Marking probability at the RED buffer at time t.
- min_{th}, max_{th}: Minimum and maximum thresholds respectively of a RED buffer.
- L: Buffer size of RED buffer.
- m: total number of TCP flows.
- $W_i(t)$: Window size of the i^{th} TCP flow at time t, $t \geq 0$, $i = 1, ..., m$.
- $\overline{r_i}$: Average Round Trip Time (RTT) for the i^{th} TCP flow, $i = 1, ..., m$.
- $T[1]$: Waiting time for the first marking event after the average queue size exceeds min_{th}.
- β_i: Number of window size increasing times during time $T[1]$ for the i^{th} TCP flow, $i = 1, ..., m$.
- t_0: Time when the first packet is marked at the RED buffer.
- t_1: Time when the last packet, which was sent just before the first window size reduction, arrives at the RED buffer.

For every packet arrival, the RED buffer estimates $\overline{Q}$ using the following exponential weighted moving average algorithm [7].

$$\overline{Q} \longleftarrow (1-\omega)\overline{Q} + Q(t)\omega, \tag{1}$$

The packet marking/dropping probability $p(t)$ is then calculated as follows:

$$p(t) = \begin{cases} 0, & 0 \leq \overline{Q} < min_{th} \\ \frac{(\overline{Q}-min_{th})max_p}{max_{th}-min_{th}}, & min_{th} \leq \overline{Q} \leq max_{th}. \\ 1, & max_{th} < \overline{Q} \leq L \end{cases} \tag{2}$$

B. Modeling Assumptions

We make the following assumptions regarding the RED buffer and TCP sources in our analytical model for minimizing buffer losses (in Sec. III-C).

- For small ω (as suggested in [7]), $\overline{Q}$ varies very slowly, so that consecutive packets are likely to experience the same marking probability [15].
- The random marking by ECN algorithm in flow i is described by a Poisson process with time varying rate $\lambda_i(t) = p(t)W_i(t)/r_i(t)$ [16]. Accordingly, the waiting time ($T_i[n]$) for the $n-$th marking event of flow i, which is given by $T_i[n] = \sum_{k=1}^{n} X_i(k)$, is a Gamma distributed random variable. $X_i(k)$ is the time interval between $(k-1)$ and $k-$th marking events for flow i. Specifically, the expected value of the waiting time for the first marking event is $E[T_i[1]] = 1/\lambda_i(t)$.
- All packets are of the same size (as used in [9]). The queue size is measured in packets.
- Packet drops at an ECN-capable RED buffer are due to either $Q(t) > L$ (buffer overflows) or $\overline{Q} > max_{th}$ (active packet drops at RED gateways). The discussion of packet drops due to buffer overflows is out of the scope of this paper.

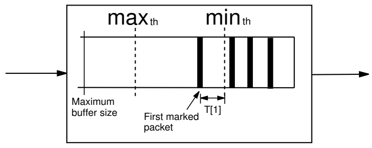

Fig. 3. Analytical model of a RED buffer.

C. Proposed model for max_{th} at RED buffers

In this section, we develop a model to estimate the optimal value of max_{th} for minimizing congestion losses at the RED buffer. We start with the recommended value of $max_{th} = 3 * min_{th}$ [7] to develop our model.

Fig. 3 shows our analytical model of a RED buffer. When the average queue size is in the steady-state condition (during which the sources are in the congestion avoidance phase), the instantaneous queue size at time t_0 is

$$Q(t_0) = min_{th} + \sum_{i=1}^{m} \beta_i, \tag{3}$$

where β_i is given by

$$\beta_i = \frac{E[T[1]]}{\overline{r_i}} = \frac{1}{\lambda_i(t)\overline{r_i}} = \frac{1}{p(t)W_i(t)}, \quad i = 1, ..., m. \tag{4}$$

Since the difference between t_0 and t_1 is one RTT, and the window size of a source is increased by one per RTT during the congestion avoidance phase, the instantaneous queue size at time t_1 can be expressed as

$$Q(t_1) = min_{th} + \sum_{i=1}^{m} (\beta_i + 1). \tag{5}$$

The average queue size is estimated using an exponential weighted moving average as shown in Eqn. (1). If time is discretized into time slots with each slot being equal to one RTT, the RED's average queue size estimation algorithm at the $k-$th slot can be expressed as

$$\overline{Q}[k+1] = (1-\omega)\overline{Q}[k] + Q[k]\omega. \tag{6}$$

Similarly, if we assume t_1 is equal to slot k in time, Eqn. (5) can be rewritten as

$$Q[k] = min_{th} + \sum_{i=1}^{m} (\beta_i + 1). \tag{7}$$

Assuming marking events in an ECN-capable RED gateway are modeled by a Poisson process, the maximum value of m can be mathematically expressed in terms of the bottleneck link bandwidth and average share of bottleneck for each TCP

TABLE II

COMPARISON BETWEEN SIMULATION RESULTS AND ANALYSIS RESULTS FOR MAXIMUM THRESHOLD (I.E., MAXIMUM AVERAGE QUEUE SIZE).

Sim. cases	ω	min_{th} (Packets)	max_{th} (Packets)	max_p	$\overline{Q}_{max}$ (Packets) Analy.	$\overline{Q}_{max}$ (Packets) Sim.
Case 1	0.002	5	15	0.1	**7.3**	**7.6**
Case 2	0.002	5	15	0.2	**6.7**	**7.1**
Case 3	0.002	7	21	0.1	**9.6**	**9.9**

flow. We have derived this mathematical expression and are currently validating our results.

In practice, ω is very small, and the congestion window size increases by one for every RTT during the congestion avoidance phase. Therefore, before the first marking event happens (i.e., no congestion control) it is reasonable to consider both the instantaneous queue size and the average queue size to be constant within a very short time period (see the first assumption in Section III-B). Thus, by plugging Eqn. (7) into Eqn. (6) and assuming that the average queue sizes during the two previous consecutive time slots are the same, the average queue size estimated at time t_1 can be solved iteratively:

$$\overline{Q}_{max} = \overline{Q} = min_{th} + \sum_{i=1}^{m} (\beta_i + \omega). \qquad (8)$$

The first marking event is followed by many random ECN marking events resulting in congestion window size adjustment of TCP sources. The average queue size stays at a level which is smaller than the average queue size at time t_1, as will be shown by our simulation results in Fig. 5 later. Therefore, **Eqn. (8) gives the maximum average queue size for minimizing congestion losses, i.e. this is our suggested value of** max_{th}.

D. Model Verification

We have simulated the topology of Fig. 2 using *ns-2* (*ns* Version 2.1b6) simulation tool from Berkeley. To make the different cases comparable, we choose RTT=59 ms for all TCP connections. The RED parameters vary depending on the case as described below.

1) Verification of max_{th}*:* We use three cases as shown in Table II to verify the validity of max_{th} as suggested by our model in Sec III-C. Case 1 uses the recommended RED parameters [7]. It is seen that $\overline{Q}_{max}$ (which we have suggested in Eqn. (8) as the value to be used for max_{th}) agrees with the value obtained from simulation.

Fig. 4 shows the congestion window size, and Fig. 5 shows instantaneousness queue size, and average queue size for Case 1. We see that the instantaneous queue size is maximum when the congestion window size reaches the slow start threshold. The average queue size is maximum just before the first marking event at time $t = 1.9$ sec. Due to space limitations, we do not show congestion window and queue sizes for Cases 2 and 3, but we have observed similar relationships.

2) Verification of Optimality of max_{th}*:* To prove that the value of max_{th} as suggested by our model is optimal in minimizing congestion losses at a RED buffer, we simulated three cases as shown in Table III. Case 1, which uses $max_{th} = 15$ as per the recommended RED parameters [7], results in

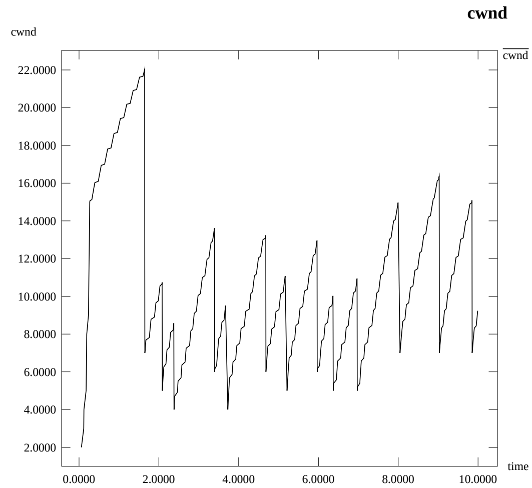

Fig. 4. Congestion window size of TCP source 1 for Case 1 in Table II.

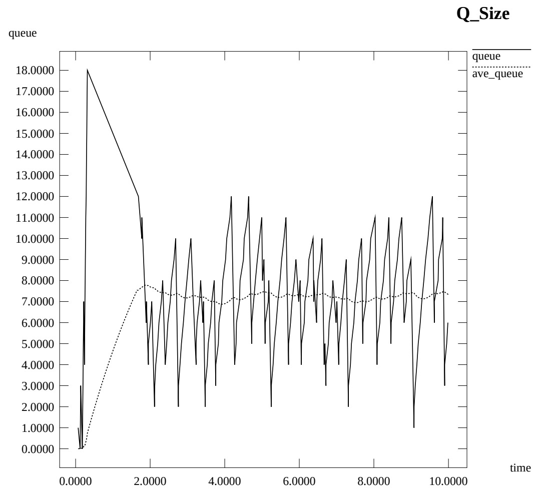

Fig. 5. Instantaneous queue size, and average queue size for Case 1 in Table II.

zero packet loss. Case 2 which uses $max_{th} = 8$ (**almost half the recommended value of 15**) as recommended by our model also results in zero packet loss. We conclude that the smaller value of max_{th} suggested by our model achieves the same congestion control effects as earlier larger recommended values. Using the value of max_{th} as suggested by our model thus results in smaller buffer size and queueing delay.

Case 3, using $max_{th} = 7$ which is one less than the value suggested by our model, results in packet loss. We therefore, conclude that **the value of** max_{th} **as obtained from our model is the minimum (optimal) value required to ensure zero packet loss at a RED buffer**.

TABLE III

PACKET DROPS WITH DIFFERENT RED PARAMETERS.

Simulation cases	ω	min_{th} (Packets)	max_{th} (Packets)	max_p	Packet drops
Case 1	0.002	5	**15**	0.1	**No**
Case 2	0.002	5	**8**	0.1	**No**
Case 3	0.002	5	**7**	0.1	**Yes**

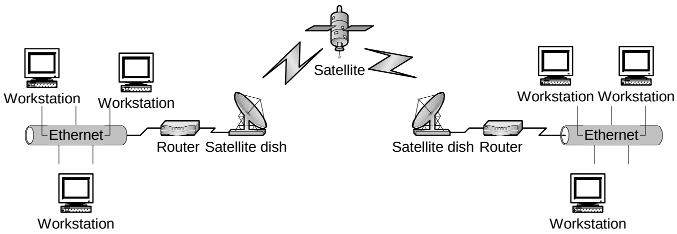

Fig. 6. LAN interconnection using a satellite link.

IV. SpecTCP Performance Evaluation

In this section, we present our simulation evaluation results. We first evaluate *SpecTCP* throughput by comparing it with TCP-Reno with ECN capability. We then present the frequency of mispredictions for *SpecTCP* in different BER environments. Finally, we compare the performance of *SpecTCP* with Snoop [17] and TCPW [18].

We have evaluated the performance of our *SpecTCP* algorithm using *ns-2*. The ECN implementation is based on RFC 2481 [14]. Our network topology for conducting simulations is shown in Figure 6. Two local area networks (10 Mbps) are connected using a satellite link (64Kbps) with a propagation delay of 280ms. A uniform random error model is used to generate random errors on the wireless link. Instead of dropping packets at routers, RED routers are used in our simulations to set the CE bit in the packet header.

The full-duplex link between router A and router B has a BER, which varies between $1e^{-7}$ to $1.2e^{-4}$ in our simulation. The receiver's advertised window size, which is also equal to the initial *ssthresh* at the sender, is set to 30 segments. The packet size is set to 1000 bytes (when BER is below $5e^{-5}$) or 512 bytes (when BER exceeds $5e^{-5}$) as explained below.

Ftp was used in our simulation to transfer data from the source to destination. We used a method called dynamic test to make the throughput measurement easier to control. Instead of fixing the size of the file to be transferred, we controlled the simulation time as a function of the BER. By changing three parameters, viz, the number of packets being dropped, simulation time and packet size, we could carry out simulations for a wide range of BER values. This resulted in a flexible and efficient way to avoid a lot of difficulties in carrying out simulations under high BER conditions. In the next section, we present results regarding the effectiveness of our algorithm.

A. Evaluation of SpecTCP Throughput

Throughput and goodput (the amount of useful information, in *bit*, being received by the receiver per second, not including errors) obtained from simulation experiments for both *SpecTCP* and TCP-Reno with ECN capability are compared.

We used the network parameters the dynamic test method to measure the *goodput* (bit/s) and the *normalized throughput* of both TCPs with BER values ranging from $1e^{-7}$ to $1.2e^{-4}$.

Fig. 7 compares the goodput in bit/s of both *SpecTCP* and TCP-Reno with ECN capability. The normalized throughput of both the TCPs are shown in Figure 8. We see that *SpecTCP*'s throughput is much higher than that of TCP-Reno with ECN. At a BER of $5e^{-5}$, the goodput of our *SpecTCP* is almost *5 times* higher than that of TCP-Reno with ECN. From Figures 7 and 8, this improvement is much higher at

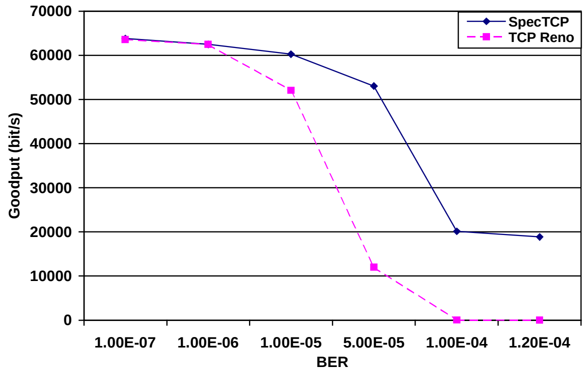

Fig. 7. Comparison of goodput (bit/s).

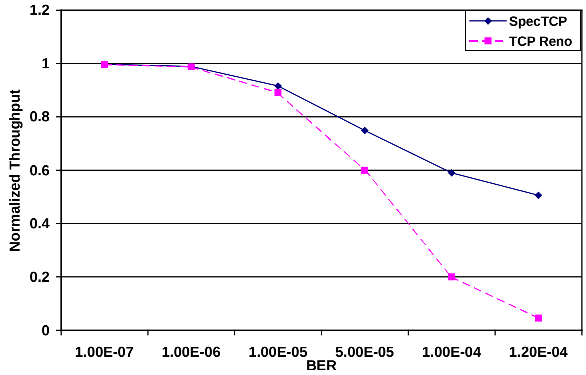

Fig. 8. Comparison of normalized throughput.

higher BER. In addition, the throughput of TCP-Reno with ECN suffers more severely than our *SpecTCP* as the error rate increases. We can also see that, with the increase of BER, the throughput of TCP-Reno with ECN decreases much faster than our *SpecTCP*. For example, according to Figure 7, when BER increases from $1e^{-5}$ to $5e^{-5}$, TCP-Reno's goodput decreases by **77 %** in contrast to our *SpecTCP* whose goodput only decreases by **12 %**. This is also valid for normalized throughput as shown in Fig. 8. This is because, at higher BER, congestion window reductions for TCP-Reno is so frequent that congestion window size cannot reach a high value.

B. SpecTCP Misprediction Rate

SpecTCP speculates that losses are due to link errors. Though we have shown the effectiveness of our model to improve the speculation accuracy, there must be mispredictions in reality. Misprediction rate is defined as the frequency when *SpecTCP* mispredicts a congestion loss as a loss due to link errors. The simulation results are shown in Fig. 9. As shown, with our analytical model to guarantee the speculation accuracy, *SpecTCP* speculates accurately, especially in high BER settings. Misprediction rates are a little higher at smaller BER values (i.e., $1e^{-7}$ and $1e^{-6}$), but they still reflect similar performances as those speculation algorithms used by computer architects [5].

C. Performance Comparison with Prior Arts

We compared the performance of *SpecTCP* with TCP with ECN, Snoop, and TCPW (TCP-Westwood). Simulation results are shown in Fig. 10 and Fig. 11.

Fig. 10 compares the congestion loss rate (the ratio of congestion losses to the total number of transmitted packets) among *SpecTCP*, TCP with ECN, Snoop, and TCPW.

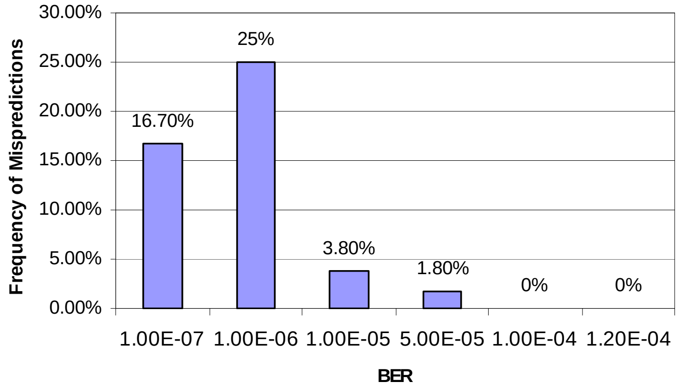

Fig. 9. Misprediction rate for SpecTCP.

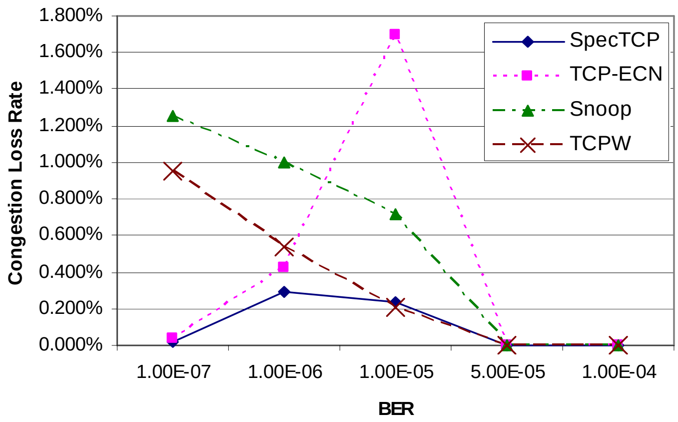

Fig. 10. Comparison of congestion loss rate.

SpecTCP has the smallest congestion loss rate. This is due to the deployment of ECN based congestion control and our congestion loss minimization model (used to guarantee speculation accuracy).

Fig. 11 compares the normalized Goodput (Goodput in bps divided by the total number of transmitted packets) among four TCP improvement algorithms. As shown in Fig. 11, *SpecTCP* outperforms other three algorithms.

V. Conclusion

We have developed an analytical model to estimate the optimal value of maximum threshold of a RED buffer in order to minimize packet losses due to congestion at RED gateways. We have shown that the analytical model matches our simulation results very well. We used congestion loss minimization model as a way to improve speculation accuracy for a new speculation-based TCP algorithm, *SpecTCP*, to improve TCP throughput in the presence of non-congestion related losses in a lossy wireless environment. With the improved speculation accuracy, *SpecTCP* improves the network throughput by speculating that loss events are caused by link errors, unless the network congestion is explicitly indicated by the receipt of ECN_ECHO packets.

The proposed *SpecTCP* has been found to significantly improve TCP performance in lossy wireless links. We have achieved significant throughput improvement, up to *five times*, over the TCP-Reno with ECN for data transfer across a typical satellite link with high bit-error rates. Results have also shown that, with ECN and congestion loss minimization model, *SpecTCP* outperforms Snoop and TCP-Westwood which do not require ECN-capable RED gateways and end systems.

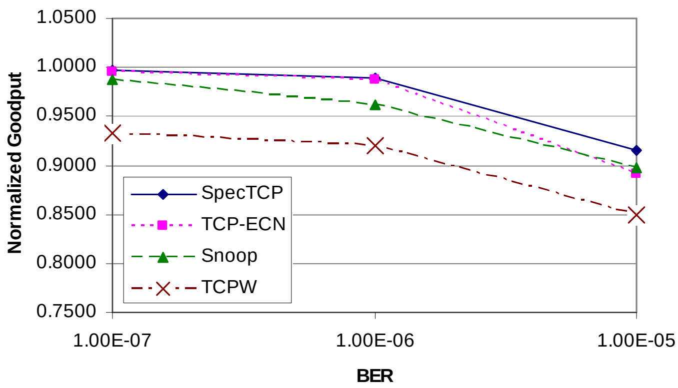

Fig. 11. Comparison of normalized Goodput.

References

[1] H. Bai and M. Atiquzzaman, "Error modeling schemes for fading channels in wireless communications: A survey," *IEEE Communications Surveys and Tutorials*, vol. 5, no. 2, pp. 2–9, October 2003.

[2] S. Dawkins, G. Montenegro, M. Kojo, V. Magret, and N. Vaidya, "End-to-end performance implications of links with errors," RFC 3155, August 2001.

[3] S. Cen, P. C. Cosman, and G. M. Voelker, "End-to-end differentiation of congestion and wireless losses," *IEEE/ACM Transactions on Networking*, vol. 11, no. 5, pp. 703–717, October 2003.

[4] Y. Chen, R. Sendag, and D. Lilja, "Using incorrect speculation to prefetch data in a concurrent multithreaded processor," in *17th International Parallel and Distributed Processing Symposium*, Nice, France, April 2003.

[5] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, 3rd ed. Morgan Kaufmann, 2003.

[6] S. Floyd, "TCP and explicit congestion notification," *ACM Computer Communication Review*, vol. 24, no. 5, pp. 10–23, October 1994.

[7] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transaction on Networking*, vol. 1, pp. 397–413, August 1993.

[8] B. Zheng and M. Atiquzzaman, "Active queue management in TCP/IP networks," in *High Performance TCP/IP Networking: Concepts, Issues, and Solutions*, M. Hassan and R. Jain, Eds. Prentice-Hall, 2004, pp. 281–307.

[9] C. Liu and R. Jain, "Improving explicit congestion notification with the mark-front strategy," *Computer Networks*, vol. 35, no. 2-3, pp. 185–201, February 2001.

[10] S. Kunniyur and R. Srikant, "End-to-end congestion control schemes: Utility functions, random losses and ECN marks," *IEEE/ACM Transactions on Networking*, vol. 11, no. 5, pp. 689–702, October 2003.

[11] H. Bai and M. Atiquzzaman, "Using ECN marks to improve TCP performance over lossy links," in *Proc. First International Conference on E-Business and Telecommunication Networks*, Setubal, Portugal, August 2004, pp. 437–445.

[12] V. Misra, W. Gong, and D. Towlsey, "Fluid-based analysis of a network of AQM routers supporting TCP flows with an application to RED," in *Proceedings of ACM SIGCOMM*, Stockholm, Sweden, September 2000, pp. 151–160.

[13] K. K. Ramakrishnan, S. Floyd, and D. Black, "The addition of Explicit Congestion Notification (ECN) to IP," RFC 3168, September 2001.

[14] K. Ramakrishnan and S. Floyd, "A proposal to add Explicit Congestion Notification (ECN) to IP," RFC 2481, January 1999.

[15] T. Bonald, M. May, and J. Bolot, "Analytic evaluation of RED performance," in *INFOCOM*, Tel-Aviv, Israel, March 2000, pp. 1415–1424.

[16] V. Misra, W. Gong, and D. Towsley, "Fluid-based analysis of a network of AQM routers supporting TCP flows with an applicaiton to RED," in *ACM SIGCOMM*, Stockholm, Sweden, 2000, pp. 151–160.

[17] H. Balakrishnan, S. Seshan, and R. Katz, "Improving reliable transport and handoff performance in cellular wireless networks," *ACM Wireless Networks*, vol. 1, no. 4, pp. 469–481, December 1995.

[18] C. casetti, M. Gerla, S. Lee, M. Sanadidi, and R. Wang, "TCP Westwood: Bandwidth estimation for enhanced transport over wireless links," in *Proceedings of ACM Mobicom*, Rome, Italy, July 2001, pp. 287–297.